A Distributed and Multi-Tiered Software Architecture for Assessing e-Commerce Recommendations

L. Palopoli[1], D. Rosaci [2*] and G.M.L. Sarné[3]

¡[1]*University of Calabria, DIMES Department. palopoli@dimes.unical.it)*
[2]*University of Reggio Calabria, DIIES Department. domenico.rosaci@unirc.it*
[3]*University of Reggio Calabria, DICEAM Department. sarne@unirc.it*

SUMMARY

Recently, an ever increasing number of e-Commerce tools has been made available that are able help customers by generating purposed recommendations. Many of them are centralized so that they have to face problems related to efficiency and scalability. A few of them are distributed, but in this case the complexity of the e-Commerce process implies computation overhead on the client side, which is often unsuitable if mobile devices are used by customers. In this paper, we study how the software distribution in recommender systems affects their performances, depending on the characteristics of the e-Commerce population. To this end, we present a distributed testbed architecture for e-Commerce recommender systems using a multi-tiered agent-based approach to generate effective recommendations without requiring such an onerous amount of computation per single client. We use such a testbed to study the main advantages and limitations associated with the problem of distributing the computation of recommendations. Copyright © 2010 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Consumer behavior; Distributed computing; Electronic commerce; Multi-agent systems; Recommender systems

1. INTRODUCTION

The use of recommender systems for supporting e-Commerce activities is rapidly increasing, due to the overwhelming amount of information that needs to be handled into nowadays Web marketplaces [35]. Recommender systems [33, 6, 7] indeed represent a much useful technology for helping customers and sellers to improve their activities on the Web [28, 10]. On the other hand, these systems are expensive solutions when applied to modern Business-to-Customer (B2C) e-Commerce processes [5], mainly due to the large size of e-Commerce communities and to the increasing use of mobile devices with relatively limited computational resources.

The use of distributed architectures to efficiently implement recommender systems has been recently investigated [3, 27, 8].

In this scenario, the architecture of the distributed system is generally composed of software agents acting as personal assistants of the human users, capable of capturing the utility function of the user preferences and making purchases on a users behalf. We refer to such a scenario in the rest of the paper.

It has been shown that the adoption of these distributed architecture introduces significant advantages on system efficiency [23], also when adopted for e-Commerce applications [17]. In

*Correspondence to: Via Graziella, Loc. Feo di Vito, 89122 Reggio Calabria, Italy

particular, it is noticeable that the main drawbacks of distributed systems, i.e. the lack of a global state and the presence of message delays, do not affect the recommendation activity. Indeed, in these cases, a global knowledge about the state of the system is not generally required and message delay is not a relevant problem because the communication model is of the peer-to-peer kind.

However, it is not clear to what extent these advantages depend on the characteristics of the e-Commerce community [16] and, more generally speaking, what are the actual limits of the adoption of a distributed approach in a B2C e-Commerce scenario. This paper is meant to provide an answer to this question.

In order to understand why such an analysis is important, consider a case where we want to design a recommender system for supporting an e-Commerce community such as, for example, a portal for sellers and customers of antiquarian products. In this case, a possible solution consists in adopting an agent-based system such that a software agent is associated with each seller and each customer to support them by generating recommendations. Notice that, in an e-Commerce scenario, there are many activities to support. To illustrate, customers have to be provided with recommendations about the antiquarian products they will be most probably interested in. Also, they will need suggestions about the most convenient merchants. Moreover, some negotiation activities between customers and merchants might be necessary, as well as a support for the delivery process and for the evaluation service. The question is: from the efficiency viewpoint, is it better either to design a single software agent assisting its user (either a customer or a merchant) in all those different activities, or to provide each user with a set of agents, where each agent is dedicated to a specific activity? In other words, given for granted that various actors are assumed to be associated to different agents (horizontal distribution), it must be decided if having a single agent per actor is more convenient than distributing the activities associated to the single actor over a set of several more specialized agents associated with that actor (vertical distribution). On the one hand, realizing a recommender featuring just one agent for each actor leads to a relatively simple system structure, whereas its complexity increases with the other choice. On the other hand, the solution with a single agent per actor leads to have more complex components, since each agent has to manage all the e-Commerce activities of the user. The question is thus: which of these two solutions is more efficient? In this paper, we will show that the answer to this question depends on some characteristics of the e-Commerce community. These characteristics include: the number of active customers, the number of merchants and the distribution of the overhead on the merchant population. Therefore, our study provides the recommender system engineer with hopefully useful information to support her/him in the design process. It is important to highlight that, in this paper, we do not deal with the issue of the recommendation effectiveness and therefore we assume that the recommender algorithm used in the system is given. Counterwisely, we discuss the two alternative ways to implement the recommender algorithm as highlighted above.

1.1. Weaknesses of current recommender systems for e-Commerce and necessity of a distributed approach

Several existing recommender systems, as those present in several e-Commerce Web sites, are based on centralized architectures [4, 20]. This means that the recommender operates on the server side by performing all the activities necessary for generating recommendations. Usually, such implementations rely on relatively simple (and, as such, not particularly effective) recommendation mechanisms. Counterwisely, implementing more sophisticated recommendation techniques on these centralized architectures generates efficiency and scalability problems on the central server. Moreover, a centralized approach is implied to be rather "closed", that is, it hardly allows the possibility for new heterogeneous components to be added to the system through time. A further relevant weakness in centralized systems is that they imply a centralization of personal information that can be viewed as a problem on the customer's privacy side. Some of the recommender approaches proposed in the literature, however, exploit distribution and this choice represents a solution for introducing efficiency [27], robustness [19] and even good privacy levels in the system. However, in most of these approaches, the complexity of the e-Commerce process results in a

DOI: 10.1002/cpe

computation overhead on the client side. Such an overhead might become easily unbearable in cases such as with access performed though mobile devices [24].

1.2. Our contribution: A study of the efficiency of B2C activities using a distributed recommender architecture

In this paper, we shall study how the performances of a distributed recommender approach depend on the characteristics of the e-Commerce community which it is applied to. In particular, we consider as relevant population characteristics: (*i*) the number of active customers, (*ii*) the number of merchants and (*iii*) the distribution of the overhead on the merchant population. In order to realize this study, we next introduce a reference distributed framework called **M**ulti **A**gent **R**ecommender for **E-C**ommerce (**MARE-C**), based on a multi-tiered agent system architecture. The multi-tiered structure allows to increase the level of computation distribution in the system, leaving the possibility to compute effective recommendations without generating such an onerous amount of computation burden on individual client sides. This reference framework embeds some general characteristics of the B2C process [21], as those described in [17], and also exploits some ideas underlying some recent distributed architectures for recommender systems [26]. In this respect, this is not to be considered a new proposal of recommender system architecture, but simply a reasonable framework to support our analysis. Few studies [1] are available in the literature that discuss the advantages, in terms of efficiency, of using distributed architectures for recommender systems [36]. However, to the best of our knowledge, no architecture is described to perform efficiency analysis in the sense we outlined above, allowing the analyst to generate simple simulations of e-Commerce environments.

In this paper, we used our framework to perform some simulations of distributed e-Commerce systems and to analyze how the advantages of the distributed computation are related to the above mentioned characteristics of the e-Commerce environment (in terms of number of active customers and so on), which has also included simulating the behaviour of an e-Commerce community in different e-Commerce scenarios.

Our experiments showed that a distributed approach introduces significant advantages in those e-Commerce scenarios characterized by a large overhead in customer requests which are much computationally demanding on the merchant side in traditional architectures. Experiments also showed that, as expected, when merchants are present in large numbers, computational overhead automatically distributes on available merchant sites and, therefore, no additional advantage is obtained. We note that the use of a realistic distributed recommender system framework, although not a particularly sophisticated one, guarantees the significance of our results.

It is important to highlight that our paper does not intend to present a new approach to recommender system design, but introduces a framework for evaluating the impact of the multi-threading software distribution in an e-Commerce recommender system. Moreover, our purpose is not that of merely assessing the advantage of a multi-threaded approach as opposed to a centralized one (that would be obvious in general and not particularly significant for the case of a recommender system). Instead, our contribution relies on the analysis of how the advantages of the distributed computation are related to the characteristics of the e-Commerce environment. For instance, we analyzed how the performances of the recommender varies when the overhead increases, fixing the number of merchants. Moreover, we studied how the the performances vary when the number of merchants increases, for a fixed overhead.

These experiments confirmed the general intuition that a multi-threaded solution is advantageous with respect to a serial one. Also, and much more interestingly, experiments deliver information about the shape of the curves (depicted in Figure 6 and 7) plotting the advantage of adopting a multi-threaded approach w.r.t. two crucial parameters, namely the number of CBB events and the amount of overhead (average number of requests to merchant servers) with respect to the number of merchants. These results clarify that the advantage of multi-threading becomes significant only if the number of CBB events is very high and only if the overhead is relevant.

More in detail, we discovered the existence of interesting thresholds for the above parameters (marking the boundary between situations where a small advantage is obtained via multi-threading

DOI: 10.1002/cpe

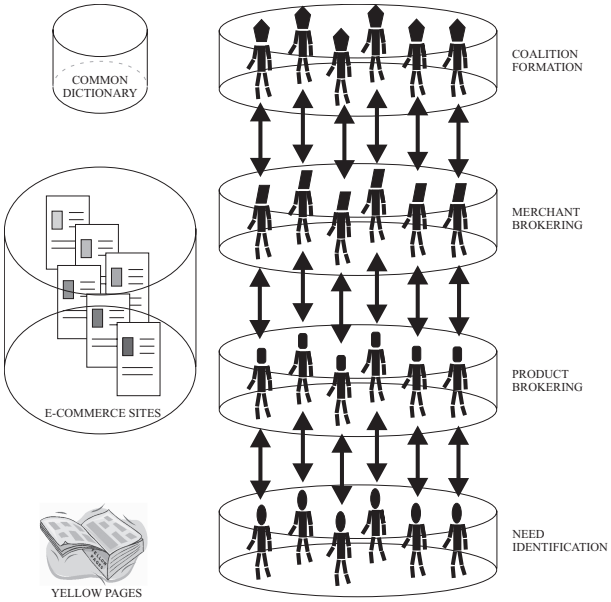

Figure 1. The multi-tiered MARE-C architecture.

to others where multi-threading becomes sharply advantageous), which we shall discuss in the following. As far as we can tell, these results are indeed new and our realistic simulations also provide actual values for these thresholds.

It is also important to observe that, in our approach, each customer and each merchant performs independently from the others, each of them collecting only the information that can be obtained by observing the behaviors of his/her interlocutors. This way, for instance, the agents of a customer collect only the information retrieved by the activities of that customer, such as the products he/she acquired, the merchants he/she interacted with and so on. Therefore, when a customer accesses a merchant site for the first time, the merchant is supposed not to already possess any information on that customer so that the search of suitable products is made autonomously by the customer's agents. In other words, only when the merchant's agents have had occasion to observe customer behavior, they had the possibility to store appropriate information on their site. Thus, this approach does not require for customers to explicitly communicate private information to merchants; rather, useful information are revealed to the merchant only through observing the customers' public behavior.

The remaining of the paper is organized as follows. Sections 3 and 4 describe the MARE-C framework data representation and the behavior, respectively. Some related work is examined in Section 5. In Section 6, a discussion is developed about the advantages and limitations theoretically obtainable by adopting distributed approaches for generating recommendations. In Section 7, we present some experiments carried out by using our MARE-C reference architecture to assess the performances of a distributed recommender system solution as opposed to a centralized one. Finally, in Section 8, some conclusions are drawn.

2. OVERVIEW

In this Section, we provide a brief overview of the MARE-C framework. As represented in Figure 1, in this framework, each customer is assisted by four software agents, each of which deals with a different stage of the CBB model [17]. More in detail:

- an NI agent assists the customer in the Need Identification stage
- a PB agent manages the Product Brokering stage
- an MB agent handles the Merchant Brokering stage

DOI: 10.1002/cpe

- a CM agent deals with the Coalition Formation stage.

This way, the overall customer e-Commerce activity is distributed over these four agents, with each agent possibly operating independently of the others and running in a different thread in the customer's machine, and having the multi-threading support on this machine to improve the efficiency of the overall process.

Each customer's agent, during his/her activity, can interact with merchant sites distributed over the Internet, where each seller is assisted by a seller agent, which internally stores both a *product catalog* containing the products offered in the site, and a *customers' profile* encoding the preferences of each customer that visited the site in the past. This interaction allows to generate content-based recommendations for the customer and also makes the site capable to generate personalized presentations of the products for its visitors, in order to support the Site Visiting stage.

The main features of this reference architecture can be, thus, summarized as follows:

- The four types of agents run on different agent *tiers*, as shown in Figure 1. Each NI (resp. PB, MB, CM)-agent can interact both with the seller agents and with the other NI (resp. PB, MB, CM)-agents, in order to generate collaborative filtering recommendations. The basic idea is that of *specializing* the interactions between agents of different types. This way, if a customer c_1 needs to interact with a customer c_2 for need identification purposes, c_1's NI-agent will interact with c_2's NI-agent with all the other agents of both c_1 and c_2 not being involved in this interaction and free to be exploited in other CBB activities.
- Each customer agent can also interact with the customer agents that manage the other stages. For example, the NI-agent of the customer c can interact with the PB-agent of the same agent c (but not with the PB-agents of the other customers) - this way, in particular, the results of a CBB stage are passed forth to the agents that manage the following stages.
- Customer and seller agents use in their internal representations of the products, *dictionaries* of product categories characterized by category names and possible relationships holding between categories. To build these personal dictionaries, a COMMON dictionary is exploited, accounting for some general information that are shared among different customers.

3. THE MARE-C KNOWLEDGE REPRESENTATION

As just pointed out, in MARE-C, all customers and sellers share a common dictionary, containing the names of basic product categories of interest for the community, as well as relationships possibly holding among them. Moreover, each agent associated to a customer has its own profile encoding the information (about the customer) the agent needs in order to manage specific CBB stage it is associated with. Similarly, each seller is assisted by a seller agent that manages his/her catalog of products, containing all offered products, organized in categories. Finally, in order to allow a collaboration between agents, a "yellow page" data structure is exploited, recording the interests that the agents desire to make public. Below we describe all these structures. Preliminarily, we introduce the concept of *Category Dictionary*, that is used as a common data structure.

A *Category Dictionary* $\mathcal{D}$ contains: (*i*) a set $\mathcal{D}_\mathcal{C}$ of *product categories* and (*ii*) a set of *category links* $\mathcal{D}_\mathcal{R}$ between categories. A category link between two categories cat_1, $cat_2 \in \mathcal{D}$ is a tuple $\langle cat_1, cat_2, t\rangle$, where t is the *type* of the link. In the current version of the MARE-C framework, we have four types of category links, namely *isa*, *synonymy*, *overlap* and *commercial* types. More in particular:

- Categories cat_1 and cat_2 are *isa-linked*, denoted $\langle cat_1, cat_2, ISA\rangle$, if all the products belonging to cat_1 also belong to cat_2. For instance, two categories such as *Old Books* and *Books* might be linked by an isa-link.
- Categories cat_1 and cat_2 are *synonymy-linked*, denoted $\langle cat_1, cat_2, SYN\rangle$, if the products belonging to cat_1 coincide with those included in cat_2. For instance, two categories such as *Multimedia* and *Audio & Video* might be considered as synonymy-linked.

DOI: 10.1002/cpe

- Categories cat_1 and cat_2 are *overlap-linked*, denoted $\langle cat_1, cat_2, OVE\rangle$, if there exists some products of cat_1 that also belong to cat_2, and vice versa. For instance, the two categories *Antiques* and *Collectibles* might be linked by an overlap-link.
- Categories cat_1 and cat_2 are *commercial-linked* (we denote with $\langle cat_1, cat_2, COM\rangle$ this relationship) if it is supposed that there are customers who purchase products belonging to both categories cat_1 and cat_2. For instance, two categories such as *Music* and *Movies* might be seen as commercial-linked.

Note that the first three types of considered category links are purely *semantic*, and also well studied in linguistics, while the *commercial* link has been introduced as a specific case of link meaningful to the e-Commerce context. We also observe that it is natural to represent a category dictionary $\mathcal{D}$ as a direct, labeled graph $G_{\mathcal{D}} = \langle \mathcal{D}_C, \mathcal{D}_R\rangle$. This graph has a (labeled) node called cat for each category $cat \in \mathcal{D}_C$, and a label denoted $info_{cat}$ is associated to this node.

We say that a category cat *belongs* to the category dictionary $\mathcal{D}$, denoted by $cat \in \mathcal{D}$, if there exists a node named cat in $\mathcal{D}$. Moreover, an arc labeled t is present in the graph for each link $\langle cat_1, cat_2, t\rangle \in \mathcal{D}_R$, oriented from the node cat_1 to the node cat_2. Note that in correspondence of synonymy, overlap and commercial links, two arcs connect the involved nodes, since these links denote bidirectional relations.

In the remaining of the paper, we shall refer to the graph $G(\mathcal{D})$ as the data structure for representing a category dictionary $\mathcal{D}$. Thus, when referring to a category dictionary $\mathcal{D}$, we will mean the graph $G(\mathcal{D})$.

Now, we can use the notion of category dictionary to define some *relationships* between categories, that we call *category relationships*. These relationships are the natural extension of the notion of link, in the following sense. If two categories cat_1 and cat_2 are t-linked in a category dictionary, and cat_2 and cat_3 are also t-linked in the same dictionary, then cat_1 and cat_3 should be considered as belonging to a more general t-relationship, with t being ISA, SYN, OVE or COM. In other words, two categories cat_1 and cat_2 are considered as *t-related* if in the dictionary $\mathcal{D}$, there is a path between the nodes associated with cat_1 and cat_2, consisting of t-labelled arcs only. We denote by ISA($\mathcal{D}$) (resp. SYN($\mathcal{D}$), OVE($\mathcal{D}$), COM($\mathcal{D}$)) the set of the pairs (cat_1, cat_2) such that cat_1 and cat_2 are ISA-(resp. SYN,OVE,COM) related. A further extension of the notion of category relationship that we introduce is that of *general* relationship, whereby two categories cat_1 and cat_2 are *generally related* if there exists, in the dictionary $\mathcal{D}$, a path from cat_1 to cat_2, independently of the specific arc-labels therein. We denote by GEN($\mathcal{D}$) the set of the pairs (cat_1, cat_2) such that cat_1 and cat_2 are generally related.

A *Common Dictionary* COMMON is assumed to be publicly available to all the customers and the sellers of the community.

3.1. The Personal Profiles

In a MARE-C community, each customer $c \in \mathcal{C}$ is assisted by four agents, each of which handles a different stage of the B2C process. In particular, as described in the Introduction, the Need Identification, Product Brokering, Merchant Brokering and Buyer Coalition Formation stages of customer c are managed by agents called NI_c, PB_c, MB_c and CM_c, respectively. Each of these agents stores an internal customer profile, which is useful to handle the corresponding stage. For the sake of uniformity, we have implemented all these profiles using category dictionaries. Thus, the agent NI_c (resp, PB_c, MB_c, CM_c) stores the profile $\mathcal{P}(NI_c)$ (resp. $\mathcal{P}(PB_c)$, $\mathcal{P}(MB_c)$, $\mathcal{P}(CM_c)$). The nodes of $\mathcal{P}(NI_c)$ (resp. $\mathcal{P}(PB_c)$, $\mathcal{P}(MB_c)$, $\mathcal{P}(CM_c)$) represent categories of interest for customer c (resp. product categories within which customer c is currently trying to find a product, product categories for which customer c is currently trying to find a merchant, product categories for which c is currently trying to purchase products *in coalition with other customers*). In each profile, a category may belong to the set of categories contained in the common dictionary. This way, the agents of the community share some common knowledge. However, in order to make the community open, we also allow for some categories present in a NI-profile to not belong to the common dictionary, being rather defined as a "personal" customer category. In this case, in order to make the personal category understandable to the other agents of the community, we impose the

DOI: 10.1002/cpe

constraint that this category must be in a general relationship with at least one category belonging to the common dictionary.

Note that the categories represented in the PB-Profile are generally a subset of the categories contained in the NI-Profile. Each arc of the PB-Profile is a copy of the corresponding arc belonging to the NI-Profile. Similarly, the categories represented in the MB-profile (resp. CM-Profile) are generally a subset of the categories contained in the PB-Profile (resp. MB-Profile) and each arc of the MB-Profile (resp. CM-Profile) is obtained by copying the corresponding arc occurring in the PB-Profile (MB-Profile).

Recall that, in each profile, a node label is associated with each category cat. We represent the information associated with category cat of the profile $\mathcal{P}(NI_c)$ (resp. $\mathcal{P}(PB_c)$, $\mathcal{P}(MB_c)$, $\mathcal{P}(CM_c)$) by the tuple $NI_c(cat)$ (resp. $PB_c(cat)$, $MB_c(cat)$, $CM_c(cat)$). To illustrate, $NI_c(cat) = \langle i, mode \rangle$, where i, a real value in [0,1], represents the *interest value* of customer c for category cat, with 0 denoting minimum interest. The field $mode$ represents the type of visibility that c desires to be associated to his/her interest for cat. In the current implementation of MARE-C, possible values of $mode$ are $public$, which means that all other customers can view this interest, and $private$, which makes this interest invisible to the other customers. From hereafter we will use the notation $t.f$ for denoting field f of tuple t.

The node label associated with the category $cat \in \mathcal{P}(PB_c)$ is a pair $PB_c(cat) = \langle i, prod \rangle$, where $PB_c(cat).i$ is the interest value for cat, "inherited" by the corresponding node in the NI-profile, while $PB_c(cat).prod$ contains a list of products of interest for customer c. These products have a real number in $[0, 1]$ associated with each of them, measuring such interest. Finally, for each product, the customer can define a list of *constraints* he/she wants to impose to the product purchase. In particular, the following constraints can be set: (*i*) the payment method must belong to the set of payment methods $payment$; (*ii*) the format of the offer associated with the product must have a specific form (e.g. an auction, or a traditional fixed-price purchase).

To denote the element of the list $prod$ associated with product p, the tuple $PB_c(cat).prod(p) = \langle i, payment, format \rangle$ is used, such that i is the interest value for the product, $payment$ (resp. $format$) is the set of payment typologies (resp. offer formats) decided for the product. The field $payment$ is a set of strings denoting payment such as "Paypal" or "bank-transfer" while, the field $format$ is a format type for the offer.

The node label of the MB-Profile is denoted by $MB_c(cat) = \langle i, prod, sellers \rangle$, where i is the interest value for category cat, "inherited" from the corresponding node in the PB-profile, while $prod$ contains a list of products of interest, in the category cat, for which c activated a merchant brokering stage, where each product is weighted by the interest value "inherited" from the PB-Profile. Finally, $sellers$ is a list of merchants selling products in category cat, where each seller weighted by a score in the real interval [0,1]. This score tells to what extent c considers that seller a "good" seller. Our weighting mechanism follows a very common approach used for computing trust and reputation measures, by exploiting coefficient normalized in $[0..1]$ as, for instance, it is described in [22]. The element of this list associated with seller s is $MB_c(cat).sellers(s) = \langle numT, ev \rangle$, where $numT$ is the number of the transactions customer c carried out in the past with seller s and ev denotes c's *personal evaluation* for that seller. We assume that the elements in the list $MB_c(cat).sellers$ are ordered in descendant score order.

Also, a node label associated with each category $cat \in \mathcal{P}(CM_c)$, represents the information related to that category necessary in the coalition formation stage. Thus, the node label is a tuple $CM_c(cat) = \langle friends, prodSell \rangle$. Here, the field $friends$ is a list of customers which c joined in the past to form a coalition. Each customer occurring in $friends$ is weighted by a real value in [0,1], which represents how much c considers that agent suitable for a coalition. We use the notation $CM_c(cat).friends(f) = \langle numC, ev \rangle$, such that $numC$ is the number of coalitions the customer formed with friend f in the past and ev is the c's *personal evaluation* for that friend.

Finally, the field $prodSell$ is a a list of elements, each of which is associated with a product p and a seller s, such that c activated a coalition formation stage to purchase p from the seller s.

DOI: 10.1002/cpe

Each element of $prodSell$ consists, in its turn, in a list of customers, and it is associated with a product-seller pair, that is, the list of customers considered suitable to form a coalition for purchasing product p from seller s (represented by $CM_c(cat).prodSell(p, s)$).

3.2. The Site Profile

In MARE-C, each seller $s \in \mathcal{S}$ is associated with a seller agent that stores a site profile containing both (*i*) a catalog of the products and (*ii*) some information about the preferences of the customers that visited the site. As for the customer's profiles, the site profile stores a category dictionary, that allows to organize the products by category. The profile of the site s, denoted by SP_s, is a category dictionary whose nodes represent product categories in which the seller s offers some products and whose arcs represent relationships between categories. For the categories present in the product catalog, we assume the validity of the same conditions that hold for the customer's NI-Profile.

A node label is associated with each category $cat \in SP_s$, denoting information useful for the seller to manage customer visits. We denote this label as $SP_s(cat) = \langle prod, customers \rangle$, where $prod$ contains a list of products that the seller s offers in category cat. To access the elements of this list, the notation $SP_s(cat).prod(p) = \langle price, payment, format \rangle$ is used, such that $price$ is the price of the product, $payment$ is the set of payment methods available for the product and $format$ is the format of purchasing available for the product (e.g., "auction" or "buy-it-now"). We assume that $price$ represents the fixed price of the product, if the format of the offer is "buy-it-now", while it represents the reserve price of the product if the format of the offer is "auction".

The element $customers$ is a list of customers interested in products of category cat. For each customer this list stores all site's products in which the customer is interested. To denote the element of this list associated with customer c we use the tuple $SP_s(cat).customer(c) = \langle numV, prod \rangle$, such that $numV$ is the number of visits the customer paid to s and $prod$ is a list of products present in the site, i.e., it is a sub-list of $SP_s(cat).prod$.

3.3. The Yellow Pages

The Yellow Pages YP is a data structure that allows customers of the MARE-C community to publish their interests, in order to facilitate mutual collaboration. YP is a set of category dictionaries $\{YP_c\}$, one for each customer c, where YP_c is a sub-graph of c's NI-profile $\mathcal{P}(CM_c)$, containing those categories $cat \in \mathcal{P}(NI_c)$ such that $NI_c(cat).i = public$.

4. AGENT'S BEHAVIOR IN MARE-C

This section aims at giving a description about the functionalities of the MARE-C framework that we shall exploit during the study of efficiency problems when generating recommendations. Therefore, its purpose is strictly technical, allowing the reader to understand the meaning of the several parameters involved in the framework.

Customer c uses his/her own client for visiting the e-Commerce sites of the MARE-C framework. This client has two main functionalities: (*i*) supporting customer navigation through the Web sites, with a normal browser; (*ii*) providing recommendations to the customer. The client interface is provided with two tabs associated with those functionalities and called *Browser* and *Recommender*, respectively, which we will describe in the next two sub-sections. It is important to point out that if customer c is a newcomer running the client for the first time, the NI-agent asks him/her to specify the categories which he/she is interested in, in order to build an initial version of the NI-Profile $\mathcal{P}(NI_c)$. To this purpose, the NI-agent downloads the last version of the shared dictionary COMMON and shows its content to c. The customer can select whatever category present in the dictionary, and the selected categories are added to $\mathcal{P}(NI_c)$; if there exists a relationship in COMMON between two categories added to $\mathcal{P}(NI_c)$, then this relationship is also added to $\mathcal{P}(NI_c)$. Moreover, the customer can add to $\mathcal{P}(NI_c)$ also one or more personal categories cat, not occurring in COMMON, by specifying both the name and a path in $\mathcal{P}(NI_c)$ joining cat with at least one

DOI: 10.1002/cpe

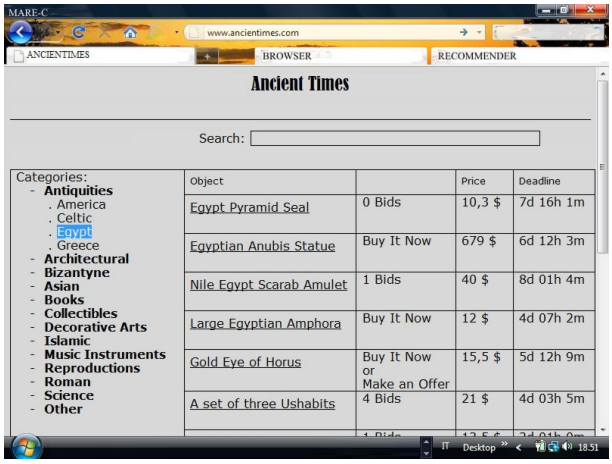

Figure 2. An example of a MARE-C site.

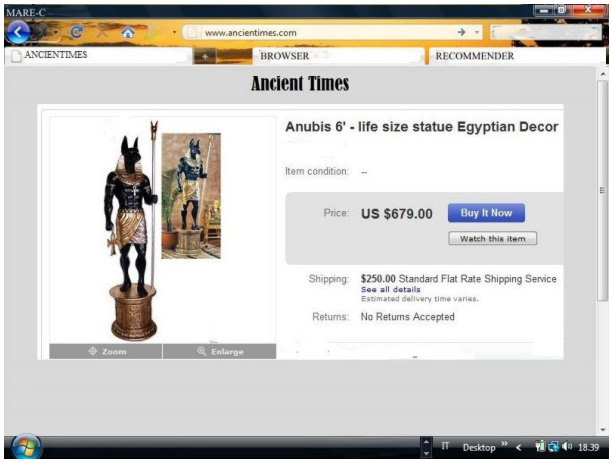

Figure 3. The visualization of the product.

category $cat^* \in COMMON$. Moreover, for each selected category, the customer is asked to specify the interest degree $NI_c(cat).i$ and the visibility mode $NI_c(cat).mode$.

4.1. The Browser: Agents working "over the shoulders"

Each MARE-C site has the eBay-like aspect shown in Figure 2; through its elements, it provides the customer with the possibility to "navigate" through the categories. Note that the site is managed by a unique merchant and shows only the products of that merchant. For instance, the site in Figure 2 deals with antiquarian products, which is apparent by considering the categories present in the list *Categories* on the left of the site page, where a category may have, in turn, some sub-categories. The customer accessing the site can visualize all the products of each category by using that list. For example, Figure 2 shows the case of a customer that is browsing the products in the sub-category *Egypt* of the category *Antiquity*. Alternatively, it is also possible for the customer to use the *Search* tool to perform a keyword-based search among the products. The list of products is visible on the right, in a 4-column table. The first column *Object* contains the name of the offered product, in the second column either the number of bids for the product (in the case the offer is an auction) or the type of the offer is shown. In this latter case, the offer can be *Buy-It-Now* (i.e., the price is fixed) or *Make an Offer* (i.e. the customer can make whatever offers he/she considers appropriate for the product and the seller can either accept or refuse them).

DOI: 10.1002/cpe

After having obtained a visualization of the list of products, the customer can perform the following actions, commonly available in most of the existing e-Commerce sites:

- **A1. Select the product**, to examine the offer in detail. To this end, the customer has to click on the name of the offer, and a frame as in Figure 3 will appear.
- **A2. Watch the product**, by clicking on the button *Watch this item* in the frame shown in Figure 3.
- **A3. Purchase the product**, that means, for a Buy-It-Now offer, as that in Figure 3, to immediately buy the product accepting the fixed price proposed by the seller. In the case of an auction, or a Make-An-Offer, purchasing a product means to make a bid or make an offer for it, respectively.

Each action performed by the customer implies an implicit call to the agents NI, PB, MB and CM, so that the profiles of these agents are automatically updated without any explicit interaction guided by the customer.

Thus, assume that customer c performs an action a of type A1, A2 or A3 for a product $p \in cat$. As a consequence, the following actions are performed by the customer's client:

- The agent NI is called, feeding it with the category cat. The following two situations can alternatively occur:
  - If cat is not present in the NI-profile, then it is added to it by the NI-agent, which also sets the interest value $NI_c(cat)$ to an initial value $iniInt$. The NI-agent, in turn, calls the agents PB, MB and CM, asking them to add cat to their profiles and set the associated interest value to the value $iniInt$.
  - Otherwise (cat already occurs in the NI-profile), the interest value of cat is updated by the NI agent, as follows:

    $$NI_c(cat) = min(1, NI_c(cat) + \Delta_a)$$

    where Δ_a is a parameter whose real value, ranging in [0,1], is added to $NI_c(cat)$ as a consequence of the performed action. The customer can choose personalized values for Δ_a, for each $a = A1, A2, A3$. As an example, the customer can set $\Delta_{A1} = 0.1$, $\Delta_{A2} = 0.15$, $\Delta_{A3} = 0.30$ to produce the effect of significantly increasing the interest value when the product has been purchased, whereas the interest increase consequent to a selection or a visit is smaller.
    The value $NI_c(cat)$ is then passed by the agent NI to the agents PB, MB and CM to be copied in $PB_c(cat).i$, $MB_c(cat).i$ and $CB_c(cat).i$, respectively.
- The client calls the PB-agent passing the product p to it. The following two situations can alternatively occur:
  - If p is not present in the PB-profile, then it is added by PB to the list $PB_c(cat).prod$ with an initial interest value set equal to $iniInt$.
    Then, the agent PB calls the agents MB and CM, which will add p to the lists $MB_c(cat).prod$ and $CM_c(cat).prod$, respectively, with interest value set equal to $iniInt$.
  - Otherwise, (p is already present in the PB-profile), then the agent PB updates the interest value of p as follows:

    $$PB_c(cat).prod(p).i = min(1, PB_c(cat).prod(p).i + \Delta_a)$$

    and then PB passes $PB_c(cat).prod(p).i$ to the agents MB and CM to be copied in $MB_c(cat).p.i$ and $CM_c(cat).prod(p).i$.
- Finally, the client calls the MB agent, passing seller s to it. Also in this case, two situations can occur:

DOI: 10.1002/cpe

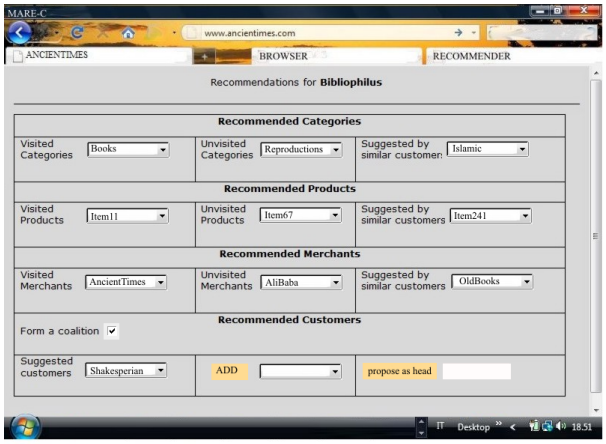

Figure 4. The recommendations provided by the personal agents.

- If seller s, the owner of the site, is not present in the MB-profile, then it is added to the list $MB_c(cat).sellers$, with an initial score equal to $iniInt$ and an initial $numT$ equal to 1. Moreover, MB calls the CM-agent, which will add s to the list $CM_c(cat).sellers$, with a score equal to $iniInt$.
- Otherwise, the MB-agent updates the score of seller s, as follows:

$$MB_c(cat).sellers(s).ev = (1, MB_c(cat).sellers(s).ev + \Delta_a)$$

and then calls the CM-agent which will copy this value in $CM_c(cat).sellers(s).ev$.

Periodically, the interest values $NI_c(cat).i$ and $PB_c(cat).prod(p).i$, as well as the evaluation of $MB_c(cat).sellers(s).ev$ are decremented by the NI-agent, PB-agent and MB-agent, respectively, to take into account the time variable. Instead, each of these agents internally stores the timestamp of the last update of the interest values and merchant scores, thus they are able to decrease these values when a given time period has elapsed. The length of the time period (called τ and represented in minutes), as well as the decrement factor (a coefficient called ρ and ranging in [0..1]), is set by the customer when configuring his/her client. In particular, we call ρ_{NI}, ρ_{PB} and ρ_{MB} (resp. τ_{NI}, τ_{PB} and τ_{MB}) those decrements (resp. the periods) associated with the NI-profile, PB-profile and MB-profile, respectively. For example, when a time period τ_{NI} has passed from the last update of the interest value $NI_c(cat).i$, the NI-agent sets $NI_c(cat).i = \rho_{NI} \cdot NI_c(cat).i$.

As for the customer's agents, also seller s performs its own activity, in order to update the site profile SP_s. For each action performed by customer c that involves a product $p \in cat$ (i.e. selecting, watching or buying p), the seller's agent updates the list $SP_s(cat).customers$. In particular, if c does not belong to $SP_s(cat).customers$, then a new element $SP_s(cat).customers(c)$ is added to the list. Moreover, product p is added to the list $SP_s(cat).customers(c).prod$ and the number of transactions $SP_s(cat).customers(c).numT$ is increased. Viceversa, if c belongs to $SP_s(cat).customers$, s updates $SP_s(cat).customers(c)$ by increasing the corresponding number of transactions $SP_s(cat).customers(c).numT$ and inserting, if not present, product p in the list $SP_s(cat).customers(c).prod$.

Note that, in our approach, each agent only stores the information that is necessary to handle its own stage.

4.2. The Recommender: Exploiting Customer's Profiles

Each agent of a customer c may generate some recommendations. The customer can examine the generated recommendations by selecting the tab "Recommender" in his/her client. As shown in Figure 4, the client visualizes a page featuring four sections, one for each of the supported stages. For each section, the recommendations are generated by the corresponding agent, by

DOI: 10.1002/cpe

following the mechanism that we shall describe below. The recommendations are generated in a "cascade" mode, that is, first the customer chooses a category from the set of categories in section "Recommended Categories"; there, for the selected category, a set of products is suggested in section "Recommended Products". When the customer chooses a product from this latter set, a set of merchants selling that product is suggested in the section "Recommended Merchants" and, at the same time, a set of friends available to form a coalition for acquiring the selected product is suggested in section "Recommended Friends" (we remark that MARE-C, in its current version, does not consider coalition for multiple-product purchases). The customer can select from the list of friends the customers with whom he/she desires to form a coalition, and his/her coalition manager interacts with a global coalition coordinator to make the coalition formation automatic. Therefore, the research of the customers having similar interests and tastes, performed by the coalition manager agents, is the basis for leading the coalition formation, since the customer chooses his/her partners from those customers that have been recognized as friends. Obviously, the customer performs his/her selection among his/her friends also based on the results obtained in past collaborations, using the assigned score. The rating is applied only to friends, so that the formation of the coalition depends both on the results of the past coalitions and on the mutual similarities of the users.

In the implementation of MARE-C used for performing the experiments, we have exploited a simplified version of the recommendation algorithm we already proposed in a previous work [25], where devices are not considered. The details of the algorithm are reported in the Appendix. We note that it is possible to use in MARE-C any other recommendation algorithm, since the distributed architecture of the framework is independent from the chosen technique.

5. RELATED WORK

Several papers have proposed solutions to the efficiency problems stemming from high dimensionality and sparsity in recommender systems. This is a critical aspect for the most common collaborative filtering algorithms that, to reach a satisfactory accuracy level, can require a large amount of time. In this context, several authors carried out experimental campaigns to analyze the figures of needed computational resources. For example, in [31] a latent semantic indexing recommender is proposed, that is able to generate a recommendation in half the time as opposed to the most common collaborative filtering recommenders. In [11], three different approaches exploiting transitive association among consumers are compared in terms of accuracy and efficiency with respect to their execution time. In [18], an Incremental Collaborative Filtering (ICF) algorithm is adopted to address the scalability problem based on incremental updates of pre-computed user-to-user similarities. A comparative test has been carried out in [37] on different collaborative filtering algorithms exploiting social tagging systems. Finally, a conversational recommender system able to help users by suggesting certain products and using users' feedbacks to refine future suggestions is proposed in [30]. In this case, the efficiency tests consist in measuring the average session length needed to generate an effective recommendation. Finally, in some recommenders [11, 18, 30, 31, 37], a particular attention was paid to optimize the efficiency by specific algorithms and verify their performances through experiments.

While most of the approaches found in literature deal with dimensionality and sparsity problems, none of them provides a testbed framework to study the efficiency of the recommender system in e-Commerce environments, by including all the stages of the B2C model. Moreover, they do not allow to compare a distributed solution to a centralized approach. The MARE-C framework we present here is suitable to be used as a testbed framework to provide such a possibility.

To make MARE-C sufficiently realistic, we have included in its features the main characteristic of distributed recommender systems proposed in the past. Although the purpose of this work is not the comparison of MARE-C (that is only a testbed framework) with real e-Commerce recommenders, we recall below some approaches to distributed recommender system design described in the literature, that we have considered when designing MARE-C.

A distributed Competitive Attention-space System (CASy) to recommend shops and bids in competitive markets, is presented in [2]. CASy is based on adaptive learning agents associated

DOI: 10.1002/cpe

with each shop that execute the task of waking up the attention of costumers. Each agent processes a large number of shop transactions and exploits information about customer interests stored in a profile, keywords, product queries and also gathering information from other shopping agents. As a consequence, the commercial strategy is efficient and adaptive in tracking the costumer for proposing suitable suggestions.

A multi-agent system implementing a knowledge-based DSR applied to the tourism domain is discussed in [14]. In this work, the agents are cooperative and recommend travel packages to the user by reciprocally exchanging information extracted from their local knowledge bases. A recommendation request is decomposed into sub-tasks handled by different agents, each maintaining its own knowledge base and working as an expert in order to compose the final recommendation. Agents autonomously select the most suitable tasks to deal with on the basis of their past experiences in a particular kind of service (hotel, flights, interchanges, conferences, etc). Specialization increases the agent's confidence and improves the quality of suggestions.

Adaptive distributed collaborative filtering recommenders implemented by means of multi-agent architectures over a P2P networks for a mobile commerce scenario are proposed in [12] and [32]. The first paper describes PEOR (PEer-Oriented Recommender system), a system devoted to generate recommendations about multimedia contents. Peers are interconnected by personal recommender agents which reside on each peer's computer and locally process suggestions. PEOR strategy, which lacks a centralized control, allows to (*i*) find neighbour peers with similar preferences, (*ii*) implement a dynamic neighbour re-formation to take into account changes in user interests, (*iii*) search for relevant and updated contents by using a rating-based filtering method based on the user's recent observation, (*iv*) suggest suitable contents to peers in real time with an event-driven push approach and (*v*) minimize the burden reminding on personal computing appliances, yet obtaining qualified recommendations. The latter paper proposes a DRS which suggests products and services for marketplace users provided with mobile devices. This recommender translates the recommendation task in a search task over a P2P topology like *Gnutella*. Peers are represented as software assistant agents of mobile customers.

In [3] a new distributed recommender system is introduced based on a user-based filtering algorithm. The model has been transposed for Peer-to-Peer architectures. It has been especially designed to deal with problems like scalability and privacy. Moreover, it adapts its prediction computations to the density of the user neighborhood. Large-scale distributed recommendation involves the coordination of various heterogeneous resources located at different nodes that need to be represented uniformly and organized into the resource space so that a semantically interactive environment can be formed.

In [13] the Knowledge Grid Based Intelligent Electronic Recommender Systems is discussed and the service oriented architecture of its knowledge grid is presented. The recommendation task is defined as a knowledge based work-flow and the knowledge grid is exploited as the platform for knowledge sharing and knowledge services which provides the functions of knowledge discovery, knowledge fusion and knowledge based work-flow definition.

In [34] a DRS is proposed consisting of multiple recommender systems from different organizations. The authors introduce a peer selection algorithm that allows a recommender system peer to select a set of other peers to cooperate with. The paper shows how it is possible to provide a solution to the problem providing effective recommendations in situations where some resources may be missing. Another distributed collaborative filtering recommender is illustrated in [29] that works over mobile ad-hoc networks. Devices use ad-hoc connections to exchange information without accessing any remote on-line directory services when, for some different reasons (e.g. cost, failure of wireless network, etc.), other communication channels are not available. The proposed system, called MobHinter, allows the identification of affinity in the neighbours (representing a narrow portion of the users' community) for obtaining personalized suggestions by the way of direct meetings among users. User affinities are modeled via a similarity graph. The collected information is then used incrementally to locally refine predictions.

Two distributed recommender systems implementing a collaborative filtering approach and having good computational efficiency are presented in [36] and [1]. Both these systems have, on

DOI: 10.1002/cpe

average, a time complexity of $\theta(n + m \cdot log_2 m)$, where n is the total number of products and m is the total number of users in the system, while the computational complexity of such class of recommenders usually lies between $\theta(m + n)$ and $\theta(m^2 n)$ [18]. By using a Distributed Hash Table (DHT) approach, the first system [36] , called DCFKQ, solves the problem of finding information in order to generate suitable recommendations of products that can meet users interests. The system of [1] instead implements a highly scalable distributed algorithm based on techniques developed in the context of auction systems like eBay [9]. In fact, it exploits a peer-to-peer solution based on a "best-seller lists" and can also tolerate malicious users (in the form of Byzantine adversary), an important advantage in the distributed peer-to-peer context.

Another common feature in the aforementioned distributed approaches is that none of them consider all the CBB stages. Differently, the testbed framework MARE-C provides both content-based and collaborative filtering recommendations and, in addition, it manages them separately for all the CBB stages. Since our paper is not addressed to make a comparison with other recommender systems from the viewpoint of effectiveness, we have highlighted that any of the previous framework considers together all the stages of the e-Commerce activities, making them not suitable to be used as frameworks for a study of efficiency. We point out that the above discussion has the only purpose of justifying the MARE-C adoption as opposed to some other systems, as the testbed system serving the purposes of our analysis. Indeed, a contribution of our paper is to provide a complete framework exploited as an effective testbed in the context of the multi agent recommenders supporting the entire CBB model and analyzed w.r.t. their distribution model, while the other frameworks mentioned above could well play this role for particular efficiency problems as, for instance, dimensionality and sparsity.

6. ADVANTAGES AND LIMITATIONS

In this Section, we discuss the practical advantages, in terms of efficiency, obtained by a multi-threading distributed approach vs a mono-thread centralized one. Also, we discuss some limitations of the multi-threading techniques, as applied to the application domain of interest. Thus, let n and m be the number of the customers and the sellers, respectively, occurring in the e-Commerce community. Moreover, let n_1 (resp. n_2, n_3 and n_4) be the number of operations needed for a user to manage the Need Identification (resp. Product Brokering, Merchant Brokering and Coalition Formation) phase. Finally, let k_1 (resp. k_2, k_3 and k_4) be the number of Need Identification (resp. Product Brokering, Merchant Brokering and Coalition Formation) sessions activated by a user at the same time.

If a unique, centralized agent manages all the four e-Commerce stages on the user's behalf, then the computational cost for this centralized agent is $NC = n_1 \cdot k_1 + n_2 \cdot k_2 + n_3 \cdot k_3 + n_4 \cdot k_4$. Instead, by using a distributed multi-agent approach, we can exploit the possibility, for each agent associated to a user and managing a given CBB stage, to manage at the same time more instances of that stage using different threads. As an example, assume that the product broker is managing a Product Brokering stage for a given product p. Now, suppose that at a given step of this stage, the product manager has to upload the product brokering profile and that, to this purpose, it makes a call to the storage memory. During the time necessary for the profile to be uploaded, the product manager is available to execute a thread managing a second Product Brokering activity for another product q. Obviously, the level of multi-threading depends on the particular stage, as well as on the system software used by the agent. This percentage is a value belonging to the interval [1/L,1], where 1 means no possibility to perform parallel executions, while 1/L means that L operations can be executed in parallel. Hereinafter, we shall denote by α_1, α_2, α_3 and α_4 the multi-threading degree for the stages of Need Identification, Product Brokering, Merchant Brokering and Buyer Coalition, respectively. Therefore, by using a decentralized architecture the computational cost for the CPU of the device exploited by the involved user is $ND = \alpha_1 \cdot n_1 \cdot k_1 + \alpha_2 \cdot n_2 \cdot k_2 + \alpha_3 \cdot n_3 \cdot k_3 + \alpha_4 \cdot n_4 \cdot k_4 + \beta$, where we indicate with β the computational overhead introduced in the distributed approach by the communications between the four local agents. Indeed, in a distributed recommendation system setting it is necessary to assume that there will be some overhead associated

DOI: 10.1002/cpe

with coordination activities ongoing among the agents. In fact, unlike the centralized system, distributed recommender systems are bound to have some overlap in terms of the data they process or the messaging overhead they incur for intra-agent communication.

By considering that, for each e-Commerce stage, each agent sends just one call message to the agent managing the next stage (e.g., the need identification agent calls the product broker), it obviously turns out that $\beta = k_1 + k_2 + k_3$.

Based on the description above, the improvement factor, denoted by ρ, deriving from the distribution can be expressed as follows:

$$\rho = \frac{\sum_{i=1}^{4}(\alpha_i \cdot n_i \cdot k_i) + \sum_{j=1}^{3} k_j}{\sum_{i=1}^{4}(n_i \cdot k_i)}$$

Now, if we assume, for simplicity, that $\alpha = \alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$, and that $k = k_1 = k_2 = k_3 = k_4$, the above formula reduces to $\rho = \alpha + \frac{3}{N}$, where $N = n_1 + n_2 + n_3 + n4$.

This formulation leads us to argue that, for high values of α (i.e. for small contribution of the multi-threading), in order to have a practical advantage, we should have to deal with a reasonably high number of operations. For instance, if $\alpha = 0.95$, the single agent should perform a minimum of $N = 60$ operations to begin to experience an advantage. Instead, if we can exploit a relevant multi-threading activity, i.e. in the case of small values of α, the advantage shows up for small values of N already. For instance, if $\alpha = 0.25$, we should ideally have an advantage when the agent performs a minimum of $N = 4$ operations. Therefore, when the device on which the agent is running is capable of offering a significant level of multi-threading, the advantage is immediately perceivable, regardless of the number of the performed operations.

To summarize, in Figure 5 we have reported, for different values of α, how the reduction ρ depends on the number of operations N.

The considerations above remain valid for each of the stages and the related coefficient a_i if considered on their own. However, it is worth pointing out that, in a situation in which α_1, α_2, α_3 and α_4 have different values, the role of the coefficients k_1, k_2, k_3, and k_4 is important. Each of these coefficient k_i, associated to stage i, amplify the advantage of the multi-threading, such that higher values of k_i imply a higher gain in terms of performances when using the multi-threads approach.

On the other hand, since $\beta = k_1 + k_2 + k_3$, and a too large value of β diminishes the positive impact of the multithreading, and uncontrolled increase of k_i values can produce an excessive message-passing overhead in return.

In the next section, we shall ground the above considerations on a more practical basis, by illustrating our experimental campaign.

7. EXPERIMENTS

Next, we present some experiments devoted to assessing the practical advantages and limitation of adopting a multi-threading distributed architecture in supporting customers and merchants in their B2C activities.

The experiments have been performed by using a simulator purposely conceived for testing different B2C scenarios, and publicly available on the Web [15]. The simulator is written in C++, and runs on a single Windows machine (Intel Core i7-4950HQ Processor, 6M Cache, 3.18 GHz, 8GB RAM), simulating the behavior of the different system agents by different threads.

In particular, for each considered B2C scenario, the simulator computes the average time exploited to perform a whole purchase process in either serial (T_s) or multi-threading (T_m) mode.

The considered scenarios involves: (*i*) a time interval of three hours, (*ii*) a population of customers performing purchases, whose size ranges between 500 and 3000 elements and a merchant population ranging between 5 and 10 elements. Moreover, each merchant should satisfy other requests coming from customers distinct from those considered in the experiment (i.e., that do not perform any purchases but send information requests, submit questions, etc.), that may require a significant amount of resources. To this aim, the simulator generates a certain number of overhead requests

DOI: 10.1002/cpe

per second. The overhead has been assumed to vary from 0 to 15 requests per second, each one randomly assigned to a merchant server. This overhead obviously affects the purchasing time of the active customers, particularly when a serial simulation is carried out.

The simulator evaluates the time costs spent for performing each considered CBB activity, namely: (*i*) Product or Merchant Brokering; (*ii*) Coalition Formation and Negotiation steps; (*iii*) concurrent overhead tasks running on the server. We have also considered, for each communication act, a mean delay due to the network load. In the simulation, each customer's purchase involves, for each CBB stage, a random number of events (with this term we denote any CBB action that requires a communication act occurring between a customer and a merchant or between two customers) varying between a lower bound and an upper bound values set for that CBB stage. Finally, the simulator allows to set the number of customers performing one or more purchases. The simulator uses the same parameter values to manage both the serial and the multi-threading cases.

We observe that the context switching between the different threads on the simulator has been considered when computing the time costs, and thus it does not affect the results in any way.

7.1. Results and Evaluation

To assess the multi-threading distributed approach, we realized two series of experiments, referred to as A and B in the following, both tested in multi-threading and in serial mode. Our experiments perform a simulation of the distributed architecture on a single machine, also simulating the overhead due to communication message passing, and thus reproducing a realistic situation.

The simulations involved a number of customers ranging from 500 to 3000 units with a step equal to 500. The first experiment has been devoted to analyze how the performances of the recommender vary when the overhead increases, fixing the number of merchants. The second series of experiments studies the variation of the performances when the number of merchants increases, for a fixed overhead. In experiment A, the number of merchants has been fixed to 10 and the overhead, represented by requests per second sent to the merchants' servers, assumes values ranging in $[0, 15]$. In other words, each client executes a random number of purchases, with the limitation that each merchant server has a maximum number of 15 additional requests per second. For experiment B, the overhead has been set to 7 requests per second and the number of merchants is varied from 5 to 10. For each simulation, the simulator computes the *average saved time* (AST), representing the percentage time difference for a whole purchase) process in a multi-threading and in a serial modality, respectively, that is, $AST = 100 * \left(\sum_{i=1}^{NP} (1 - T_{mi}/T_{si})/NP \right)$, where NP is the number of purchases performed in a simulation while T_s and T_m are the average time spent to perform the purchase in serial and multi-threading mode, respectively.

The results of experiment A are reported in Table I and graphically displayed in Fig. 6 and 7, while those of experiment B are reported in Table II and shown in Fig. 8 and 9. More in detail, for the scenario adopted for the respective experiments, the two tables report the AST values and the associated number of CBB events for each simulation. Fig. 6 and 8 show how the AST value depends on the overhead, the number of customers, the number of merchants and the number of CBB events. Note that the number of CBB events strictly depends on both the number of customers and on set of values adopted for the simulator parameters. Eventually, Fig. 7 and 9 highlight the close link between AST and the ratio overhead/number of merchants (O/M).

Analyzing the results, we note that all the performed simulations show significant advantages of the multi-threading solution with respect to the serial one. In fact, all the AST curves are characterized by a section that presents a stable trend of very slow growth, pointing out that the multi-threading approach spends about $42 - 44\%$ time of the serial approach for performing a purchase. Differently from this, the remaining part of the AST curves shows that the AST values vary very quickly.

This behavior is mainly due to the fact that changes in the parameters of the experiments (i.e., number of customers, overheads and so on) have a minimal impact on the performances of the multi-threading simulations in terms of T_m. Conversely, the serial simulations are characterized by a very high sensitivity to such changes and pay a high price in terms of T_s. In particular, Fig. 7 and 9 show how the AST curves: *i)* are stable when the ratio O/M is less than 1, *ii)* slowly increase when O/M

DOI: 10.1002/cpe

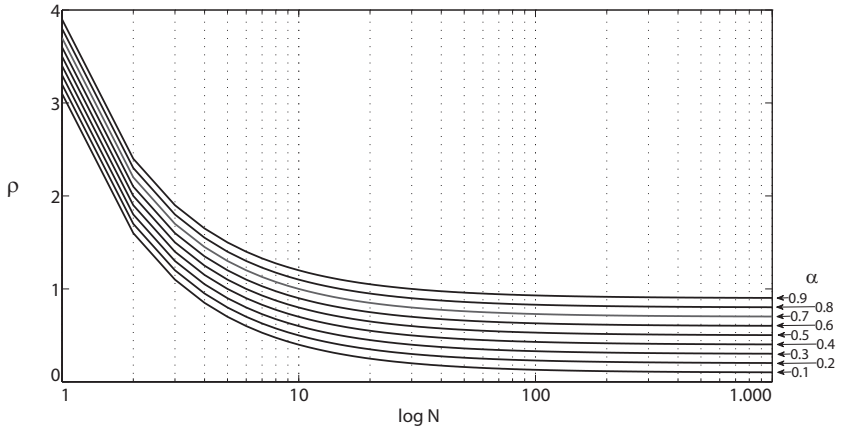

Figure 5. The variation of the reduction factor ρ depending on the number N of the local agents' operation for different values of α

Table I. Experiment A - AST measures and number of CBB events for different active Customer (C) population and 10 Merchants and for different Overheads (O).

C	Ø			
	0	1	2	3
500	42.42/ 72050	42.45/ 83272	42.87/ 95288	42.65/106088
1000	42.60/145084	42.92/157262	42.56/166330	42.34/176328
1500	42.70/217790	42.86/231270	42.93/240842	42.69/249362
2000	42.78/291506	43.12/306152	42.97/314820	42.92/323276
2500	42.99/366074	42.87/372906	42.96/384972	43,12/394680
3000	43.08/437256	43.19/449352	43.40/459172	43.58/471252
	4	5	6	7
500	42.68/116524	42.86/127222	42.28/135930	42.69/147958
1000	42.60/188632	42.67/200598	42.64/209164	43.12/222928
1500	42.76/260922	42.96/272922	43.24/284746	43.27/295860
2000	43.14/334746	43.20/344194	43.54/358308	43.38/364914
2500	43,47/407166	43,57/420884	43,60/426980	44,00/438938
3000	43.80/481774	44.03/493265	44.19/502930	44.55/511394
	8	9	10	11
500	43.06/160558	42.77/169898	46.99/179748	67.53/192844
1000	43.00/231182	43.13/241836	49.86/254270	75.77/263872
1500	43.60/306056	44.05/316714	53.33/324480	79.20/336100
2000	43.76/376788	44.28/384942	56.57/397160	79.96/411558
2500	44.36/448614	45.49/463004	58.33/471564	80,78/483438
3000	45.36/527192	46.39/532618	60.85/543794	80.41/560428
	12	13	14	15
500	77.48/201062	83.24/213240	85.28/222622	88.61/234804
1000	83.09/276146	87.01/285834	89.77/295452	91.33/308686
1500	85.58/346112	88.66/356728	91.24/369472	92.69/381380
2000	86.35/419158	89.62/433360	91.70/439258	93.12/453610
2500	86,93494144/	90,23/505374	92,48/514674	93,88/527846
3000	87.29/564578	90.64/579926	93.11/589256	94.28/592976

is close to 1 and *iii)* quickly increase for values of O/M greater than 1. This is due to the fact that the ratio $O/M \cong 1$ means that, on average, each merchant's server is busy satisfying other overhead requests. This implies, in both the experiments, that the time T_s necessary to complete a purchase in the serial approach grows with the level of "saturation" of the merchants' servers, worsening the quality of the service.

DOI: 10.1002/cpe

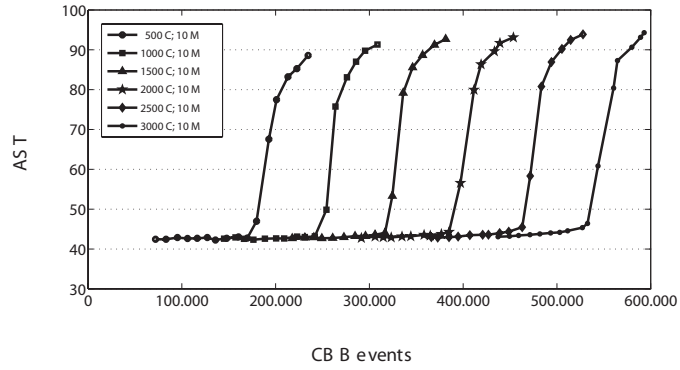

Figure 6. Experiment A - The variation of the AST depending on the number of CBB events and Overhead (O) for different active Customer (C) population and 10 Merchants (M).

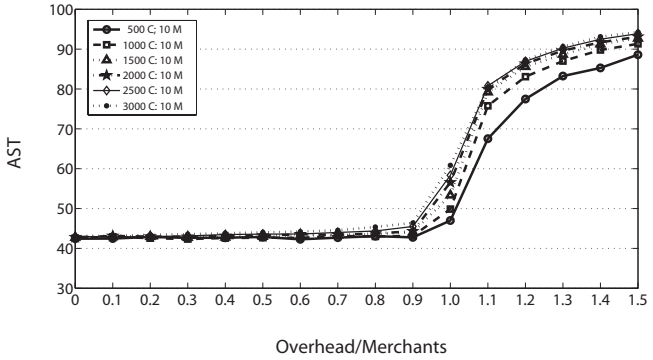

Figure 7. Experiment A - The variation of the AST depending on the ratio Overhead/number of Merchants (O/M) for different active Customer (C) population and 10 Merchants (M).

Table II. Experiment B - AST measures and number of CBB events for an overhead fixed to 7 and for different active Customer (C) population and Merchants (M).

C	M					
	5	6	7	8	9	10
500	42.71/148662	43.24/150088	42.68/147810	49.10/148534	82.75/147050	91.23/148460
1000	43.01/221628	43.12/222176	43.38/222042	55.91/220884	86.28/219622	93.07/220084
1500	43.20/292882	43.66/296130	44.36/295804	60.86/295234	87.73/294434	93.88/294996
2000	43.63/368732	44.31/372036	45.28/368292	61.24/366736	87.86/366296	94.18/370454
2500	43.94/438594	44.94/440874	46.17/437652	63.87/436646	88.39/437232	94.92/439298
3000	44.71/514144	45.65/515742	48.05/514958	66.51/516596	88.60/514852	94.95/515418

Roughly speaking, the experiment A shows that the multi-threading is advantageous, with respect to a mono-threaded solution, when we have an overhead sufficiently high for each merchant server, otherwise, in the case of a low overhead, the merchant server is able to satisfy the requests also with a mono-thread approach. Differently, the experiment B clarifies that the multi-threading is advantageous if we have a number of merchants sufficiently small with respect the number of requests, otherwise, in the case of a high number of merchants, the overhead on each merchant server would be adequately satisfied also with a mono-thread approach.

More in particular, the key result emerging from these results is that in a context where we have operations that are executed generally by merchants, operations that involve preferably only clients, and operations requiring both merchant and client activities, the best choice to improve the efficiency of the whole system consists in having a large number of clients (even if equipped with

DOI: 10.1002/cpe

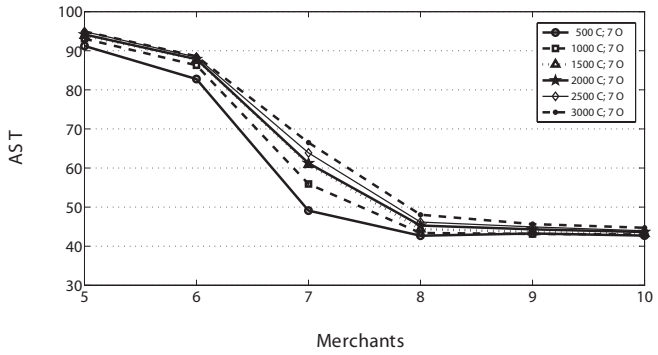

Figure 8. Experiment B - The variation of the AST depending on the number of Merchants (M) and overhead (O) for different active Customer (C) population and an Overhead (O) fixed to 7.

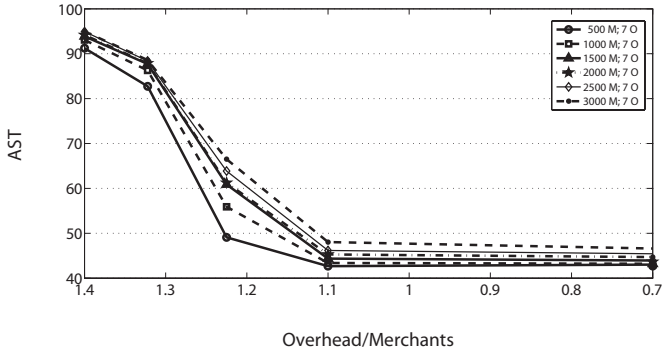

Figure 9. Experiment B - The variation of the AST depending on the ratio Overhead/number of Merchants (O/M) for different active Customer (C) population and an Overhead (O) fixed to 7.

limited resources) that contribute to some activities (e.g., forming coalition) and a small number of site merchants. Instead, the advantage strictly depends on the number of CBB events and on the overhead perceived by each merchant. If we have a large activity on the whole system (which generally depends on the fact that we have a large number of clients, even if each client has limited resources and does not individually perform many activities), then the use of a central merchant site is just a disadvantage.

In other words, the advantage of using the multi-threading becomes significant only when the number of CBB events is very large, implying the important conclusion that the adoption of a distributed approach in an e-Commerce scenario is advantageous only if the e-Commerce generates a sufficiently large activity.

The possibility for the clients to perform a limited amount of computation is certainly reasonable also for mobile clients. If the number of clients is very large, our results show that this contribution drastically improves system performances, without obliging a unique merchant site to adopt unnecessary sophisticated (and even expensive) scaling out techniques.

Another interesting consideration suggested by our experiments is related to the use of mobile devices. The experiments show that the multi-threading approach gives a significant advantage only in the presence of an intense level of activity. This result is particularly relevant if most of the clients use mobile devices. Actually, the advantage becomes significant if the number of clients is much larger than the number of merchants. In this case, it is rather probable that the ratio Overhead/Merchants is high, leading to the right sector of the Figure 7.

DOI: 10.1002/cpe

We also observe that actual recommendation systems are implemented on multi-threading in order to obtain scalability. However, for a given user, the e-Commerce stages we have considered in our model are intrinsically sequential, if performed by a central system, even provided with a multi-threading support. Obviously, each merchant server can activate different threads for different customers, but this does not influence our comparison. Also if he/she activates different threads for different customers, if he/she continues to act as a unique e-Commerce agent, he/she cannot distribute the different stages of each customer. He/She can only speed-up the single operations, by serving more customers at the same time. But this is possible also if we provide this (multi-threaded) merchant with a distributed configuration for the different stages of the e-Commerce, and providing also the customers with this analogous configuration. Such a distributed configuration will introduce a new level of the distribution, not only limited to serving more customers at the same time, but also to distribute the e-Commerce stages. Therefore, the considerations we have made in our comparison remain always valid.

Finally, as a final consideration, we highlight that, in our experiments, we have been limited to deal with only a few thousands of users, due to the fact that our simulator reproduces the distributed scenario on a unique machine, with comprehensible performance limitations. However, the obtained results clearly show that the average saved time AST asymptotically tends to reach a (high) stable value independently from the number of users and merchants (see Figure 6 and 7).

8. CONCLUSIONS AND FUTURE WORKS

In this paper, we discussed advantages and limitations of using a multi-tier distributed architecture to generate recommendations for the users of an e-Commerce community. Most available recommender systems for e-Commerce are based on centralized architectures, characterized by limitations in terms of openness, privacy and efficiency. Some proposals presented in the past adopt distributed solutions, but no analysis has been conducted in the past to provide more precise information about the advantages we intuitively expect from this type of solutions. Starting from the consideration that such an analysis can only be performed using a suitable testbed framework, we have presented in this paper the framework MARE-C, purposely conceived to test distributed recommender systems for e-Commerce. The MARE-C architecture allows to assign each stage of an e-Commerce activity to a different agent, thus creating a tier of agents associated to each stage. Using MARE-C, we have been able to test the significant reduction of the computational burden per single device on which the local agents run, as opposed to the case of a centralized architecture. Indeed, we have performed some experiments aiming at analyzing how the distributed system performances depend on some relevant characteristics of the e-Commerce community, such as the number of active customers, the number of merchants and the distribution of the overhead on the merchant population, and we have also provided a discussion about these issues.

We highlight that the contribution of our paper is not only limited to confirm by experiments the intuition that the multi-threading is beneficial when the merchant servers get saturated by their workload. The experiments we have performed are the first attempt, to the best of our knowledge, to investigate the characteristics of the advantage obtained by the multi-threading, in order to give practical indications for designing efficient e-Commerce systems. The results shown in Figures 6-9 indicates that using the multi-threading in presence of a saturation close to, but lower than 1 is much more efficient than using it in presence of a saturation close to, but higher than 1. This result was not obvious, while the intuition could suggest that, for instance, the increment of the requests increases the advantage in a significant manner also when we are under the saturation point. Figure 6 clarifies that this intuition is not true, and that the advantage under the saturation point is almost constant, drastically increasing over it

As for our ongoing research, we are currently planning to use MARE-C to study other crucial problems when using distributed solutions for recommender systems, such as reliability, dependability and security. We think that the idea of distributing the activity of the e-Commerce agents on different tiers provides an important opportunity to build an effective reputation model,

DOI: 10.1002/cpe

that could make an e-Commerce system more reliable. This will be the main direction of our future work in this area.

APPENDIX: THE RECOMMENDATION ALGORITHMS

The recommendations of the NI-agent

The NI-agent suggests to customer c the categories that can interest him/her. To this purpose, the NI-agent visualizes, in the section "Recommended Categories" present in the tab "Recommender" of the client (see Figure 4) a set of recommended categories. These categories are visualized in three distinct listboxes, called "visited categories", "unvisited categories" and "suggested by similar customers", respectively. The listbox "visited categories" contains the list of all the categories which c is supposedly interested in, thus considering the e-Commerce sites visited in the past. The listbox "unvisited categories" contains a list of categories unknown to c but which are anyway related to c's interests. Finally, the "suggested by similar customers" listbox contains categories suggested by customers having interests similar to those of c. In order to determine the content of these category lists, the NI-agent uses three approaches, namely, a content-based, a relationship-based and a collaborative filtering mechanism. More in detail:

- **Content-based**: The categories listed in the "visited categories" listbox are all those contained in the NI-profile $\mathcal{P}(NI_c)$, whose construction is performed monitoring the customer's activity, as explained in Section 4. All the interesting categories determined this way, whose set we denote by VISITED, are associated with an interest value.
- **Relationship-based**: The categories shown in the listbox "unvisited categories" are not known to the customer, but the NI-agent determined those they might be interesting to him/her. This capability of the NI-agent of proactively discovering new categories derives, in MARE-C, by a particular recommendation mechanism, called *relationship-based*, that exploits the interaction between c's NI-agent and the seller agent of each site s visited in the past by c. In particular, the seller agent of s operates as follows. For each category cat visited in the past by c (i.e. $c \in SP_s(cat).customers$), the seller agent determines all the categories $cat^* \in SP_s$ such that cat^* has not been visited by c and there exists a path in SP_s between cat and cat^*. The list of all the categories cat^* so determined is sent to c's NI-agent, that adds them to the set UNVISITED (obviously, if a category cat^* determined by s is already in UNVISITED, it is not added to the set). Note that the operations above are performed by all the seller agents of all the sites visited by c, and each seller agent of a site s performs the operations above for all the customers visiting s.
- **Collaborative-filtering**: The categories shown in the listbox "suggested by similar customers" are determined by exploiting the collaboration between c's NI-agent and the NI-agents of other customers having interests similar to those of c. To this purpose, c's NI-agent exploits the public repository YP. As described in Section 3, YP contains, for each customer x joined with the MARE-C platform, the x's public interest profile YP_x. Thus, c's NI-agent computes the similarity degree $s(c,x)$ between its NI-profile $\mathcal{P}(NI_c)$ and each YP_x, where the similarity degree is computed as the Jaccard measure of the set of nodes of $\mathcal{P}(NI_c)$ and YP_x, that is:

$$s(c,x) = \frac{|NODES(\mathcal{P}(NI_c)) \bigcap NODES(YP_x)|}{|NODES(\mathcal{P}(NI_c)) \bigcup NODES(YP_x)|}$$

  where NODES(G) returns the set of nodes of its input graph.
  This way, c's NI-agent determines the n customers that have the largest similarity value, as compared to c, where n is a parameter set by the customer. Then, c's NI-agent determines, for each similar customer x, those categories contained in the corresponding public profile YP_x that are not contained in $\mathcal{P}(NI_c)$, and adds them to the set OTHERS.

DOI: 10.1002/cpe

As shown in Figure 4, the NI-agent displays the categories belonging to the set VISITED in the listbox "visited categories", ordered by interest value, while the categories contained in UNVISITED (resp. OTHERS) are visualized in the listbox "unvisited categories" (resp. "suggested by similar customers"), ordered alphabetically.

Note that the generated recommendations are not associated to the particular site currently visited, visualized in the browser tab, but they are related to the whole customer's navigation history through e-Commerce sites.

The recommendations of the PB-agent

When a customer c clicks on a category cat occurring in section "Recommended Categories" (in the listbox "visited categories" or "unvisited categories" or "suggested by similar customers", as shown in Figure 4) on his/her client, the PB-agent is called and visualizes, in the section "product recommendations", a set of recommended products of the selected category. These products are partitioned in three sets, as for the recommended categories, namely the sets VISITED, UNVISITED and OTHERS, that are visualized in the "visited products", "unvisited products" and "suggested by similar customers" listboxes, respectively. The construction of these sets is carried out by using two different recommendation mechanisms:

- **Content-based**: The set VISITED contains all the products of the category cat that belong to the PB-profile $\mathcal{P}(PB_c)$, ordered by interest value. On the contrary, the set UNVISITED is built by exploiting a collaboration with the seller agent of each site s customer c visited in the past. In particular, the seller agent sends to c's PB agent the set of all the products of category cat that are in the site profile and that have not been visited by c in the past. All the products sent by the different seller agents are inserted in the set UNVISITED (eliminating possible duplicates).
- **Collaborative-filtering**: The second recommendation mechanism exploits the collaboration between c's PB-agent and the PB-agents of other customers having interests similar to those of c. To this purpose, c's PB-agent calls c's NI-agent to obtain the list of the n customers that share the largest values of similarity degree with c. Then, c's PB-agent calls the PB-agent of each similar customer x, to obtain the set of those products that are in the product list $PB_x(cat).prod$ and inserts them in OTHERS.

As shown in Figure 4, the PB-agent shows the products, belonging to the set VISITED in the listbox "visited products", ordered by interest value, and the products belonging to UNVISITED (resp. OTHERS) in the listbox "unvisited products" (resp. "suggested by similar customers"), ordered alphabetically.

The recommendations of the MB-agent

If customer c clicks on a product p in section "Recommended Products" (either in the listbox "visited products" or "unvisited products" or "suggested by similar customers") on his/her client, the MB-agent is called and visualizes, in section "merchant recommendations", a set of recommended sellers of the selected category. Also in this situation, as in need identification and in product brokering, the recommendations are partitioned in the three sets VISITED, UNVISITED and OTHERS (see Figure 4).

The construction of these sets is carried out by using the same recommendation mechanisms used for product brokering, that are:

- **Content-based**: The set VISITED contains the merchants associated with category cat that belong to the MB-profile $\mathcal{P}(MB_c)$, ordered by score. Instead, the set UNVISITED exploits a collaboration between the c's MB-agent and the seller agent of each site s: c's MB-agent asks the seller agent of s if category cat belongs to the site profile of s. If the seller agent of s responds affirmatively, then s is added to the UNVISITED set.

DOI: 10.1002/cpe

- **Collaborative-filtering**: To build the set OTHERS, c's MB-agent contacts the MB-agents of the n customers most similar to c. Each of these customers sends to c's MB-agent its set of merchants $MB_x(cat).sellers$, that is added to the set OTHERS.

As shown in Figure 4, the MB-agent shows the merchants belonging to the set VISITED in the listbox "visited merchants", ordered by score, and the set UNVISITED (resp. OTHERS) in the listbox "unvisited merchants" (resp. "suggested by similar customers"), ordered alphabetically.

The recommendations of the CM-agent

When customer c selects a seller s from one of the listboxes in section "Recommended Merchants" and also checks the check-box "form a coalition", the CM-agent generates a list of recommended customers that appear suitable to form a coalition for purchasing product p, belonging to category cat, from seller s. To this purpose, c's CM-agent initially inserts a new element $CM_c(cat).prodSell(p,s)$ in $CM_c(cat).prodSell$ (At the beginning, this element is equal to the empty set). Then, it sends a request to the CM-agents of all the customers belonging to the community. The request is a proposal to form a coalition for purchasing product p from seller s. Each contacted CM-agent, associated to a customer x, verifies if the element $CM_x(cat).prodSell(p,s)$ appears in its list $CM_x(cat).prodSell$. In the affirmative case, this means that also x is trying to form the same coalition as c, and then the x's CM-agent inserts c in the list $CM_x(cat).prodSell(p,s)$ and affirmatively responds to c's CM-agent request. The latter adds x to the list $CM_c(cat).prodSell(p,s)$ in $CM_c(cat).prodSell$. Finally, after a time τ (that the customer can set) from the time point in which the coalition formation attempt was started, c's CM agent visualizes in the listbox "recommended customers for a coalition" the customers stored in the list $CM_c(cat).prodSell(p,s)$. These selected customers are shown ordered by either score or number of coalitions formed with c (c can choose the ordering criterium). To this purpose, the score and number of coalitions are retrieved from the list $CM_c(cat).friends$, if the tuple $CM_c(cat).friends(x)$ belongs to this list; otherwise both these values are set equal to 0.

Then, customer c can select from this listbox the customers with whom he/she desires to form a coalition, and proposes a *coalition head* (see Figure 4), that is the customer that will make the offer on the behalf of the coalition. c's CM-agent sends a message to the Coalition Coordinator CC, containing the desired members for the coalition and the coalition head to acquire product p from merchant s. The CC determines if there exists a set of customers that mutually desires to form a coalition with each other, and assigns the role of coalition head to the customer that reported the majority of the votes. Finally, the CC sends the composition of the so created coalition to all the involved customers, and the coalition head will be thus able to make the offer. A-posteriori, each involved customer can assign an evaluation score in [0,1] to each member of the coalition. The CM-agent of each member, say c, will update the element $CM_c(cat).friends(x)$ for each customers x with whom he/she formed the coalition, by adding the new values $CM_c(cat).friends(x).numC$ and $CM_c(cat).friends(x).ev$.

Supporting the site visits

For each seller s, the associated seller agent SA_s has the role of personalizing the presentation of its site to the specific customer that is visiting it. To this purpose, the seller agent exploits the site profile SP_s. When a customer c visits the site s, the seller agent verifies if the element $SP_s(cat).customers(c)$ exists, i.e., if the customer has already visited the site in the past. In the affirmative case, the home page of the site is personalized for c, by visualizing in a "shop window" those elements belonging to the list $SP_s(cat).customers(c).prod$, that represents a sort of *local profile* of c related to s. The visualized products are ordered by interest value. At the same time, the seller agent increases the value $SP_s(cat).customers(c).numV$, for taking into account the new visit of c. In the case customer c is visiting site s for the first time, the seller agent visualizes a standard default home page.

DOI: 10.1002/cpe

REFERENCES

1. B. Awerbuch, B. Patt-Shamir, D Peleg, and M.R. Tuttle. Improved Recommendation Systems. In *Proc. of 16th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 1174–1183. SIAM, 2005.
2. S.M. Bohte, E. Gerding, and J.A. La Poutré. Market-based Recommendation: Agents that Compete for Consumer Attention. *ACM Trans. Internet Techn.*, 4(4):420–448, 2004.
3. S. Castagnos and A. Boyer. Personalized Communities in a Distributed Recommender System. In *Advances in Information Retrieval*, volume 4425 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2007.
4. P. De Meo, D. Rosaci, G.M.L. Sarnè, G. Terracina, and D. Ursino. EC-XAMAS: Supporting E-Commerce Activities by an XML-based Adaptive Multi-Agent System. *Applied Artifificial Intelligence*, 21(6):529–562, 2007.
5. H. Dong, F.K. Hussain, and E. Chang. A service concept recommendation system for enhancing the dependability of semantic service matchmakers in the service ecosystem environment. *Journal of Network and Computer Applications*, 34(2):619–631, 2011.
6. F. Buccafurri, G. Lax, D. Rosaci, D. Ursino. A user behavior-based agent for improving web usage. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* , 2519 LNCS, pp. 1168-1185, 2002.
7. P. De Meo and A. Nocera and G. Terracina and D. Ursino. Recommendation of similar users, resources and social networks in a Social Internetworking Scenario. *Information Sciences*, 181(7):1285–1305, 2011.
8. P. De Meo and G. Quattrone and D. Ursino. Integration of the HL7 Standard in a Multiagent System to Support Personalized Access to e-Health Services. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1244–1260, 2011.
9. eBay URL. http://www.ebay.com. 2011.
10. S. Garruzzo, D. Rosaci, and G.M.L. Sarné. MARS: An Agent-Based Recommender System for the Semantic Web. In *Distributed Applications and Interoperable Systems*, volume 4531 of *Lecture Notes in Computer Science*, pages 181–194. Springer Berlin Heidelberg, 2007.
11. Z. Huang, H. Chen, and D. Zeng. Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142, 2004.
12. J.K. Kim, H.K. Kim, and Y.H. Cho. A User-Oriented Contents Recommendation System in Peer-to-Peer Architecture. *Expert Systems with Applications*, (34):300–312, 2008.
13. P. Liu, G. Nie, D. Chen, and Z. Fu. The Knowledge Grid Based Intelligent Electronic Commerce Recommender Systems. In *Proc. of the. IEEE International Conference on Service-Oriented Computing and Applications, 2007 (SOCA '07), Newport Beach, CA, US, June 19-20, 2007*, pages 223–232, Washington, DC, USA, 2007. IEEE Computer Society.
14. F. Lorenzi, F.A. Correal, A.L. Bazzan, M. Abel, and F. Ricci. A Multiagent Recommender System with Task-Based Agent Specialization. In *Proc. of 10th Int. Work. on Agent Mediated Electronic Commerce (AMEC), in conjuction with AAMAS-2008, Estoril, Portugal, May 12-16, 2008*, 2008.
15. MAREC Web Site. http://marec.unirc.it, 2015.
16. L. Palopoli, D. Rosaci, and G.M.L. Sarné. A Multi-tiered Recommender System Architecture for Supporting e-Commerce. In *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 71–81. Springer, 2013.
17. L. Palopoli, D. Rosaci, and D. Ursino. Agents' roles in b2c e-Commerce. *AI Communications*, 19(2):95–126, 2006.
18. M. Papagelis, I. Rousidis, D. Plexousakis, and E. Theoharopoulos. Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms. In *Proc. of the 15th Int. Symp. on Foundations of Intelligent Systems (SMIS 2005), Saratoga Springs, NY, May 25-28, 2005*, Lecture Notes in Computer Science, pages 553–561. Springer, 2005.
19. N. Parikh and N. Sundaresan. Buzz-based Recommender System. In *Proc. of 18th Int. Conf. on World Wide Web (WWW09), Madrid, Spain, April 20-24, 2009*, pages 1231–1232, New York, NY, USA, 2009. ACM.
20. M.N. Postorino and G.M.L. Sarné. A Neural Network Hybrid Recommender System. In *Proc of the 2011 Conf. on Neural Nets WIRN10*, pages 180–187, Amsterdam, The Netherlands, 2011. IOS Press.
21. D. Rosaci. Exploiting Agent Ontologies in B2C Virtual Marketplaces. *Journal of Universal Computer Science*, 11(6):1011–1039, 2005.
22. D. Rosaci. Trust measures for competitive agents. *Knowledge-Based Systems*, 28:38–46, 2012.
23. D. Rosaci and G.M.L. Sarné. MASHA: A Multi-Agent System Handling User and Device Adaptivity of Web Sites. *User Modeling and User-Adapted Interaction*, 16(5):435–462, 2006.
24. D. Rosaci and G.M.L. Sarné. Cloning mechanisms to improve agent performances. *Journal of Network and Computer Applications*, 2012.
25. D. Rosaci and G.M.L. Sarnè. A multi-agent recommender system for supporting device adaptivity in e-Commerce. *Journal of Intelligent Information System*, 38(2):393–418, 2012.
26. D. Rosaci and G.M.L. Sarnè. Recommending Multimedia Web Services in a Multi-Device Environment. *Information Systems*, 2012.
27. D. Rosaci, G.M.L. Sarné, and S. Garruzzo. MUADDIB: A Distributed Recommender System Supporting Device Adaptivity. *ACM Transactions on Information Systems*, 27(4), 2009.
28. J.B. Schafer, J.A. Konstan, and J. Riedl. E-Commerce Recommendation Applications. *Data Min. Knowl. Discov.*, 5(1-2):115–153, 2001.
29. R. Schifanella, A. Panisson, C. Gena, and G. Ruffo. MobHinter: Epidemic Collaborative Filtering and Self-Organization in Mobile Ad-Hoc Networks. In *Proc. of 2008 ACM Conf. on Recommender Systems, RecSys 2008, Lausanne, Switzerland, October 23-25, 2008*, pages 27–34, New York, NY, USA, 2008. ACM.
30. B. Smyth, L. McGinty, J. Reilly, and K. McCarthy. Compound Critiques for Conversational Recommender Systems. In *Proc. of the IEEE/WIC/ACM Int. Conf. on Web Intelligence, Beijing, China, September 20-24, 2004*, WI '04, pages 145–151, Washington, DC, USA, 2004. IEEE Computer Society.

DOI: 10.1002/cpe

31. P. Symeonidis, A. Nanopoulos, A.N. Papadopoulos, and Y. Manolopoulos. Collaborative Recommender Systems: Combining Effectiveness and Efficiency. *Expert Systems with Applications*, 34(4):2995 – 3013, 2008.
32. A. Tveit. Peer-to-Peer based Recommendations for Mobile Commerce. In M.V. Devarakonda, A. Joshi, and M.S. Viveros, editors, *Proc. of 1st Int. Work. on Mobile Commerce, Rome, Italy, July, 21, 2001*, pages 26–29, New York, NY, USA, 2001. ACM.
33. D. Ursino, D. Rosaci, G.M.L. Sarnè, and G. Terracina. An Agent-based Approach for Managing e-Commerce Activities. *International Journal of Intelligent Systems*, 19(5), 2004.
34. L.T. Weng, Y. Xu, Y. Li, and R. Nayak. A Fair Peer Selection Algorithm for an Ecommerce-Oriented Distributed Recommender System. In *Proc. of the 4th Conf. on Advances in Intelligent Information Technology, Brisbane, AUS, June 7-9, 2006*, pages 31–37, Amsterdam, The Netherlands, 2006. IOS Press.
35. C. Yue and H. Wang. Profit-aware Overload Protection in E-commerce Web Sites. *Journal of Network and Computer Applications*, 32(2):347–356, 2009.
36. F. Zhang, S. Feng, D. Jia, and Q. Tian. DCFKQ:A DHT-based Distributed Collaborative Filtering Algorithm. *J. of Computational Information Systems*, 6(1):97–104, 2010.
37. N. Zheng, Q. Li, S. Liao, and Zhang L. Which Photo Groups Should I Choose? A Comparative Study of Recommendation Algorithms in Flickr. *J. Inf. Sci.*, 36(6):733–750, 2010.

DOI: 10.1002/cpe